Coleções .NET

Algoritmos e Estruturas de Dados

Edwaldo Soares Rodrigues

Estruturas de dados básicas

- A plataforma .NET fornece classes de **estruturas de dados** pré-empacotadas
- Essas classes são conhecidas como **classes de coleção**
- Ao utilizar essas classes, nós utilizamos as estruturas de dados sem nos preocuparmos de que maneira elas foram implementadas (*reutilização de código*)
- Necessário usar o namespace System.Collections.Generic

Estruturas de dados básicas

- Algumas coleções disponíveis na plataforma .NET
  - List<T>
  - Queue
  - Stack
  - Dictionary
  - SortedList

Estruturas de dados básicas

- Classe **List<T>**
  - Frequentemente, precisamos usar vetores sem, entretanto, saber o tamanho ideal
  - A classe *List<T>* imita um vetor convencional e, adicionalmente, fornece redimensionamento dinâmico
  - Um objeto *List<T>* pode conter uma quantidade de elementos menor ou igual à sua capacidade

Estruturas de dados básicas

- Classe **List<T>**
  - A capacidade de um *List<T>* pode ser manipulada através de sua propriedade *Capacity*
  - Se um *List<T>* precisa crescer, por padrão ele duplica sua capacidade (*Capacity*) atual
  - *Lists* representam uma lista de objetos fortemente tipada, que podem ser acessados por índice.

Estruturas de dados básicas

- Classe **List<T>**
  - Criação de um List<T> sem informar a capacidade
    - Ex: `List<int> L = new List<int>();`
  - Criação de um List<T> informando a capacidade
    - Ex: `List<int> L = new List<int>(10);`

Estruturas de dados básicas

- Classe **List<T>**
  - A capacidade (Capacity) e a quantidade de elementos (Count) inicial de um *List<T>* é ZERO
  - Após inserir o primeiro elemento, a capacidade é igual a 4
  - Exercício: faça um pequeno programa que crie um *List<T>* e imprima sua capacidade e quantidade de elementos. Adicione um elemento e repita a impressão.

Estruturas de dados básicas

- Classe **List<T>**

```
using System;
using System.Collections.Generic;

class Program{

  static void Main(string[] args){
    //List<int> AL = new List<int>();
    List<int> AL = new List<int>();
    Console.Write("AL.Capacity = {0} - AL.Count = {1}\n", AL.Capacity, AL.Count);
    AL.Add(1);
    Console.Write("AL.Capacity = {0} - AL.Count = {1}\n", AL.Capacity, AL.Count);
    AL.Add(5);
    Console.Write("AL.Capacity = {0} - AL.Count = {1}\n", AL.Capacity, AL.Count);
    AL.Add(15);
    Console.Write("AL.Capacity = {0} - AL.Count = {1}\n", AL.Capacity, AL.Count);
    AL.Add(3);
    Console.Write("AL.Capacity = {0} - AL.Count = {1}\n", AL.Capacity, AL.Count);
    AL.Add(13);
    Console.Write("AL.Capacity = {0} - AL.Count = {1}\n", AL.Capacity, AL.Count);
  }
}
```

```
> dotnet run
AL.Capacity = 0 - AL.Count = 0
AL.Capacity = 4 - AL.Count = 1
AL.Capacity = 4 - AL.Count = 2
AL.Capacity = 4 - AL.Count = 3
AL.Capacity = 4 - AL.Count = 4
AL.Capacity = 8 - AL.Count = 5
>
```

Estruturas de dados básicas

Método	Descrição
Add	Adiciona um objeto do tipo T no final do List<T>.
BinarySearch	Realiza uma busca binária e retorna a posição do objeto (a partir de 0) ou um número negativo se o objeto não for encontrado.
Clear	Remove todos os elementos da List<T>.
Contains	Retorna true se o objeto passado por parâmetro estiver em List<T>.
IndexOf	Retorna o índice da primeira ocorrência no List<T> do objeto passado por parâmetro.
Insert	Insere um objeto na posição passada por parâmetro. Ocorre uma exceção se a posição não existir.
LastIndexOf	Retorna o índice da última ocorrência no List<T> do objeto passado por parâmetro.
Remove	Remove a primeira ocorrência do objeto passado por parâmetro.
RemoveAt	Remove o objeto na posição passada por parâmetro.
RemoveRange	Remove um intervalo de elementos do List<T>.
Reverse	Inverte a ordem dos elementos do List<T>.
Sort	Ordena o *List<T>*
ToArray	Copia os elementos de um List<T> para um vetor (Array).

Estruturas de dados básicas

Propriedade	Descrição
Capacity	Obtém ou define o número total de elementos que a estrutura de dados interna pode manter sem redimensionamento.
Count	Obtém o número de elementos contidos no *List<T>*.
Item	Obtém ou define o elemento no índice especificado.

Estruturas de dados básicas

Método	Exemplo
Add	L.Add(15); L.Add(3); L.Add(5);
BinarySearch	Posicao = L.BinarySearch(3);
Clear	L.Clear()
Contains	if (L.Contains(15)) Console.Write(“Elemento encontrado”);
Count	Qtde = L.Count;
IndexOf	Pos15 = L.IndexOf(15);
Insert	L.Insert(2, 125); // Adiciona 125 na posição 2
LastIndexOf	Pos15 = L.LastIndexOf(15);
Remove	L.Remove(3); // Não ocorre exceção se elemento inexistente
RemoveAt	L.RemoveAt(1); // Ocorre exceção de posição inexistente
RemoveRange	L.RemoveRange(0, 2); // Remove 2 elementos à partir da posição 0. Ocorre exceção se não existir a quantidade de elementos desejada

Reverse	L.Reverse(); // Inverte os elementos de todo o *List*
	L.Reverse(3,5); // Inverte os 5 elementos à partir da posição 3

Estruturas de dados básicas

Método	Exemplo
Sort	L.Sort();
ToArray	int[] vetor; ... vetor = L.ToArray();

Como iterar?

- List<T>
  - Podem ser utilizados tanto os comandos de repetição for/while/do while quanto o comando foreach
  - Exemplo

```
List<int> L = new List<int>();
…
for(int i = 0; i < L.Count; i++)
   Console.WriteLine(L[i]);
foreach(int o in L)
   Console.WriteLine(o);
foreach(int num in L)
   Soma = Soma + num;
```

Como iterar?

- List<T>
  - Podem ser utilizados tanto os comandos de repetição for/while/do while quanto o comando foreach
  - Exemplo

```
List<int> L = new List<int>();
…
for(int i = 0; i < L.Count; i++)
   Console.WriteLine(A[i]);
foreach(int o in L)
   Console.WriteLine(o);
foreach(int num in L)
   Soma = Soma + num;
```

Como iterar?

- List<T>
  - Podem ser utilizados tanto os comandos de repetição for/while/do while quanto o comando foreach
  - Exemplo

```
List<int> L = new List<int>();
…
for(int i = 0; i < L.Count; i++)
   Console.WriteLine(L[i]);
foreach(int o in L)
   Console.WriteLine(o);
foreach(int num in L)
   Soma = Soma + num;
```

Como iterar?

- List<T>
  - Podem ser utilizados tanto os comandos de repetição for/while/do while quanto o comando foreach
  - Exemplo

```
List<int> L = new List<int>();
…
for(int i = 0; i < L.Count; i++)
   Console.WriteLine(L[i]);
foreach(int o in L)
   Console.WriteLine(o);
foreach(int num in L)
   Soma = Soma + num;
```

Como iterar?

- List<T>
  - Podem ser utilizados tanto os comandos de repetição for/while/do while quanto o comando foreach
  - Exemplo

```
List<int> L = new List<int>();
…
for(int i = 0; i < L.Count; i++)
   Console.WriteLine(L[i]);
foreach(int o in L)
   Console.WriteLine(o);
foreach(int num in L)
   Soma = Soma + num;
```

Estruturas de dados básicas

- Classe **Queue<T>**
  - Implementa a estrutura de dados FILA
  - Filas são estruturas do tipo FIFO (*First-in First-out*)
  - Operações básicas de uma fila : *enqueue* e *dequeue*
  - *Enqueue* recebe um objeto como argumento e o adiciona no fim da FILA (enfileiramento)
  - *Dequeue* remove e retorna o objeto que está no início da FILA

Estruturas de dados básicas

Método	Descrição
Clear	Remove todos os elementos do *Queue<T>*
Contains	Verifica se o elemento passado por parâmetro está contido em *Queue<T>*.
Dequeue	Remove e retorna o objeto do início do *Queue<T>*.
Enqueue	Adiciona um objeto no final do *Queue<T>*.
Peek	Retorna o objeto do início do Queue<T> sem removê-lo.
ToArray	Copia os elementos do *Queue<T>* para um novo array.

Propriedade	Descrição
Count	Retorna a quantidade de objetos que o *Queue<T>* REALMENTE está armazenando.

Estruturas de dados básicas

- Classe **Stack**
  - Implementa a estrutura de dados PILHA
  - Pilhas são estruturas do tipo LIFO (*Last-in First-out*)
  - Operações básicas de uma pilha : *push* e *pop*
  - *Push* recebe um objeto como argumento e o empilha no topo da pilha
  - *Pop* remove e retorna o objeto que está no topo da pilha

Estruturas de dados básicas

Método	Descrição
Clear	Remove todos os objetos de *Stack<T>*.
Contains	Verifica se o elemento passado por parâmetro está contido em *Stack<T>*.
Peek	Retorna o objeto do início do *Stack<T>* sem removê-lo.
Pop	Remove um objeto do topo do *Stack<T>*.
Push	Adiciona um objeto no topo do *Stack<T>*.
ToArray	Copia os elementos do *Stack<T>* para um novo array.

Propriedade	Descrição
Count	Retorna a quantidade de objetos que o *Stack<T>* REALMENTE está armazenando.

Como iterar?

- Queue e Stack
  - Essas classes **não podem ser iteradas como um vetor**, assim como pode ser feito com List's
  - PORTANTO, SÓ É POSSÍVEL ITERAR Queue's e Stack's por meio do comando foreach
  - Exemplo

```
Queue<int> Q = new Queue<int>();
…
for(int i = 0; i < Q.Count; i++)
   Console.WriteLine(Q[i]);
foreach(object o in Q)
   Console.WriteLine(o);
foreach(int num in Q)
   Soma = Soma + num;
```

Como iterar?

- Queue e Stack
  - Essas classes não podem ser iteradas como um vetor, assim como pode ser feito com List's
  - PORTANTO, SÓ É POSSÍVEL ITERAR Queue's e Stack's por meio do comando foreach
  - Exemplo

```
Queue<int> Q = new Queue<int>();
…
for(int i = 0; i < Q.Count; i++)
   Console.WriteLine(Q[i]);
foreach(object o in Q)
   Console.WriteLine(o);
foreach(int num in Q)
   Soma = Soma + num;
```

Como iterar?

- Queue e Stack
  - Essas classes não podem ser iteradas como um vetor, assim como pode ser feito com List's
  - PORTANTO, SÓ É POSSÍVEL ITERAR Queue's e Stack's por meio do comando foreach
  - Exemplo

```
Queue<int> Q = new Queue<int>();
…
for(int i = 0; i < Q.Count; i++)
   Console.WriteLine(Q[i]);
foreach(object o in Q)
   Console.WriteLine(o);
foreach(int num in Q)
   Soma = Soma + num;
```

Como iterar?

- Queue e Stack
  - Essas classes não podem ser iteradas como um vetor, assim como pode ser feito com List's
  - PORTANTO, SÓ É POSSÍVEL ITERAR Queue's e Stack's por meio do comando foreach
  - Exemplo

```
Queue<int> Q = new Queue<int>();
…
for(int i = 0; i < Q.Count; i++)
   Console.WriteLine(Q[i]);
foreach(object o in Q)
   Console.WriteLine(o);
foreach(int num in Q)
   Soma = Soma + num;
```

Como iterar?

- O mesmo se aplica ao Stack
  - Exemplo

```
Stack<int> S = new Stack<int>();
…
for(int i = 0; i < S.Count; i++)
   Console.WriteLine(S[i]);
foreach(object o in S)
   Console.WriteLine(o);
foreach(int num in S)
   Soma = Soma + num;
```

Erro!!!

Estruturas de dados básicas

- Classe **Dictionary<T, T>**
  - Representa uma estrutura do tipo Dicionário/Mapa composta por pares chave/valor (*key/value*) que são organizados com base no código *hash* da chave
  - Situação: Você precisa pesquisar eficientemente os dados dos 1000 funcionários de uma empresa. Se você quisesse utilizar os 11 dígitos do CPF como índice para armazenar e pesquisar tais dados, precisaria criar um vetor com 999.999.999 posições
  - O problema é que as chaves estarão espalhadas em um intervalo muito grande

Estruturas de dados básicas

- Classe **Dictionary<T, T>**
  - A solução é converter as chaves (no exemplo anterior, o CPF) em índices exclusivos no vetor, o que é a base da técnica de *hashing*
  - Uma função de *hash* efetua um cálculo para determinar a posição de determinado dado no vetor

Estruturas de dados básicas

Método	Descrição
Add	Adiciona a chave e o valor ao dicionário.
Clear	Remove todos os elementos do dicionário.
ContainsKey	Determina se o *Dictionary<TKey, TValue>* contém a chave especificada.
ContainsValue	Determina se o *Dictionary<TKey, TValue>* contém um valor específico.
Remove(TKey)	Remove o valor com a chave especificada
Remove(Tkey,Tvalue)	Remove o valor com a chave especificada do *Dictionary<TKey,TValue>* e copia o elemento para o parâmetro value.
TryGetValue	Obtém o valor associado com a chave especificada.

Estruturas de dados básicas

Propriedade	Descrição
Count	Obtém o número de pares chave/valor, contidas no *Dictionary<TKey, TValue>*.
Item	Obtém ou define o valor associado com a chave especificada.
Keys	Obtém uma coleção que contém as chaves de *Dictionary<TKey, TValue>*.
Values	Obtém uma coleção que contém os valores de *Dictionary<TKey, TValue>*.

Como iterar?

- Classe **Dictionary<T, T>**
  - Assim como Queue e Stack, objetos da classe Dictionary não podem ser iterados como um vetor;
  - PORTANTO, SÓ É POSSÍVEL ITERAR Dictionary por meio do comando foreach

Como iterar?

- **Dictionary<T, T>**

```
Dictionary<int, string> D = new Dictionary<int, string>();
…
```

Como iterar?

- Iterando sobre as chaves

```
foreach (int chave in D.Keys)
   Console.Write(chave+" ");
```

- Iterando sobre os valores

```
foreach (String valor in D.Values)
   Console.Write(valor +" ");
```

Como iterar?

- Iterando sobre as chaves e valores

```
foreach (KeyValuePair<int,string> kv in D)
   Console.WriteLine(kv.Key+"\t"+kv.Value);
```

Estruturas de dados básicas

- Classe **SortedList<T, T>**
  - Representa uma estrutura do tipo Dicionário/Mapa composta por pares chave/valor (*key/value*) ordenados pela chave e acessíveis tanto pela chave quanto pelo índice

Estruturas de dados básicas

Método	Descrição
Add	Adiciona um objeto com a chave e valor determinado
Clear	Remove todos os elementos
ContainsKey	Verifica se contém um objeto com uma chave específica
ContainsValue	Verifica se contém um objeto com um valor específico
Remove	Remove o elemento com a chave especificada
RemoveAt	Remove o elemento no índice especificado
IndexOfKey	Procura a chave especificada e retorna seu índice
IndexOfValue	Pesquisa o valor especificado e retorna o índice da primeira ocorrência
TryGetValue	Obtém o valor associado à chave especificada.

Estruturas de dados básicas

Propriedade	Descrição
Capacity	Obtém ou define o número de elementos que o *SortedList<TKey,TValue>* pode conter.
Count	Obtém o número de pares chave/valor, contidas no *SortedList<TKey, TValue>*.
Item	Obtém ou define o valor associado com a chave especificada.
Keys	Obtém uma coleção que contém as chaves de *SortedList<TKey, TValue>,* na ordem de classificação.
Values	Obtém uma coleção que contém os valores de *Dictionary<TKey, TValue>,* na ordem de classificação de sua chave.

Como iterar?

- Classe **SortedList<T, T>**
  - Assim como Queue e Stack, objetos da classe SortedList não podem ser iterados como um vetor;
  - PORTANTO, SÓ É POSSÍVEL ITERAR SortedList por meio do comando foreach

Como iterar?

- **SortedList<T, T>**

```
SortedList<int, string> SL = new SortedList<int, string>();
…
```

Como iterar?

```
// Iterando sobre as chaves - usando foreach
Console.WriteLine("\nCHAVES\n======");
foreach (Object chave in SL.Keys)
    Console.Write(chave + " ");

// Iterando sobre os valores - usando foreach
Console.WriteLine("\n\nVALORES\n=======");
foreach (Object valor in SL.Values)
    Console.WriteLine(valor + " ");

// Iterando sobre chaves e valores - usando foreach
Console.WriteLine("\nCHAVES E VALORES\n================");
Console.WriteLine("Chave\tValor");
foreach (DictionaryEntry DE in SL)
    Console.WriteLine(DE.Key + "\t" + DE.Value);
```

Estruturas de dados básicas

- Classe **LinkedList<T>**
  - Representa uma estrutura do tipo lista duplamente encadeada;
  - Estruturas encadeadas trazem maior eficiência, já que possibilitam melhor gerenciamento de memória;

Estruturas de dados básicas

Método	Descrição
AddAfter	Adiciona o valor especificado após o nó (célula) passado por parâmetro.
AddBefore	Adiciona o valor especificado antes do nó (célula) passado por parâmetro.
AddFirst	Adiciona um elemento no início do LinkedList<T>.
AddLast	Adiciona um elemento no final do LinkedList<T>.
Clear	Remove todos os elementos do LinkedList<T>.
Contains	Verifica se o elemento está contido em LinkedList<T>.
Find	Encontra o primeiro nó (célula) que possui o valor passado por parâmetro. Retorna um LinkedListNode<T>.
FindLast	Encontra o último nó (célula) que possui o valor passado por parâmetro. Retorna um LinkedListNode<T>.
Remove	Remove a primeira ocorrência do valor passado por parâmetro. Retorna false se o valor não for encontrado.
RemoveFirst	Remove o primeiro nó (célula).
RemoveLast	Remove o último nó (célula).

Estruturas de dados básicas

Propriedade	Descrição
Count	Retorna a quantidade de objetos que o List<T> REALMENTE está armazenando.
First	Retorna o primeiro nó (célula) de um LinkedListNode<T>.
Last	Retorna o último nó (célula) de um LinkedListNode<T>.

Como iterar?

- Classe **LinkedList<T>**
  - Assim como Queue e Stack, objetos da classe LinkedList não podem ser iterados como um vetor;
  - PORTANTO, SÓ É POSSÍVEL ITERAR LinkedList por meio do comando foreach

Como iterar?

- **SortedList<T, T>**

```
LinkedList<int> LL = new LinkedList<int>();
…
```

Como iterar?

```
// Iterando sobre uma Lista Duplamente encadeada
```

```
LinkedList<int> LL = new LinkedList<int>();
LL.AddFirst(10);
LinkedListNode<int> current = LL.FindLast(10);
LL.AddBefore(current, 5);
LL.AddLast(15);
current = LL.Find(15);
LL.AddAfter(current, 25);

foreach (int v in LL)
  Console.Write(v + " ");
Console.WriteLine("\n\n");
```

Bibliografia

- SHARP, John – Microsoft Visual C# 2013 passo a passo – Capítulo 18
- DEITEL, H. – C# Como Programar – Capítulo 23
- SCHILDT, Herbert – C# 4.0: The Complete Reference – Capítulo 24
- http://www.dotnetperls.com/
- http://msdn.microsoft.com/en-us/library/k166wx47_ (MSDN)